

Netcore应用

dotnet \*.dll --urls "http://\*:5001;"

dotnet \*.dll -ip 127.0.0.1 -port 5026s

dotnet WebApiCore.dll --urls=http://\*:5026 --ip=127.0.0.1 --port=5026

dotnet WebApiCore.dll --urls=http://\*:5027 --ip=127.0.0.1 --port=5027

dotnet WebApiCore.dll --urls=http://\*:5028 --ip=127.0.0.1 --port=5028

dotnet WebApiCore.dll --urls=http://\*:5029 --ip=127.0.0.1 --port=5029

dotnet WebApiProduct.dll --urls=http://\*:5126 --ip=127.0.0.1 --port=5126

dotnet WebApiProduct.dll --urls=http://\*:5127 --ip=127.0.0.1 --port=5127

dotnet WebApiCoreGateway.dll --urls=http://\*:6299 --ip=127.0.0.1 --port=6299

dotnet WebApiAuthentication.dll --urls=http://\*:7200 --ip=127.0.0.1 --port=7200

dotnet IdentityServerCore3.dll --urls=http://\*:6200 --ip=127.0.0.1 --port=6200

IdentityServerCore3.dll

Consul.exe 启动指令: Consul.exe agent -dev

Nginx启动指令：

start nginx.exe

./nginx -s stop  
./nginx -s quit  
./nginx -s reload

2.重新加载配置文件： 当 ngin x的配置文件 nginx.conf 修改后，要想让配置生效需要重启 nginx，使用-s reload不用先停止 ngin x再启动 nginx 即可将配置信息在 nginx 中生效，如下： ./nginx -s reload

dotnet WebCoreTest3.1.dll --urls=http://\*:6001 --ip=127.0.0.1 --port=6001

dotnet IdentityServerCore3.1.dll --urls=http://\*:5005--ip=127.0.0.1 --port=5005

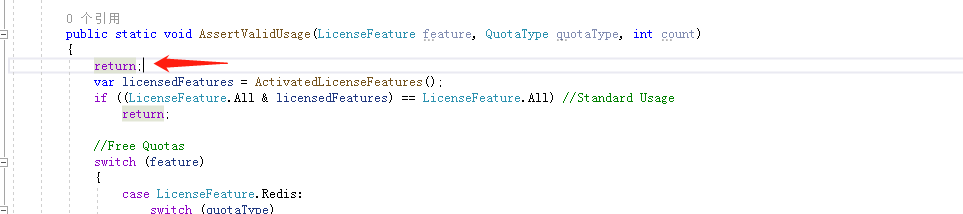
https://localhost:5001/.well-known/openid-configuration

dotnet MvcClient.dll --urls=http://\*:5006

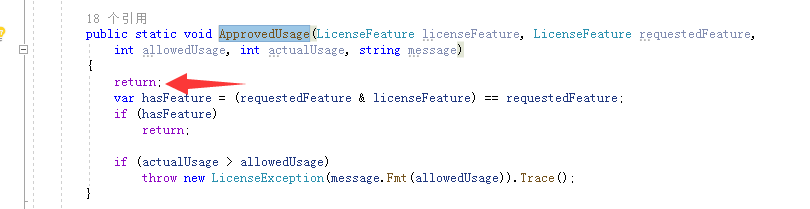
Redis 破解每小时6000次限制

第一步下载：<https://github.com/ServiceStack/ServiceStack.Text>

第二步：找到ServiceStack.Text类库下LicenseUtils.cs类文件，找到AssertValidUsage静态方法，可以直接返回return,也可以



或者直接改

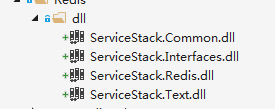


改完后编译，到bin文件夹下拿到

\bin\Debug\netstandard2.0\ServiceStack.Text.dll

第三步骤：去除项目中nuget包引用，然后医用dll，

用第二步中破解版的ServiceStack.Text.dll替换掉nuget包下载的



Redis基本指令

八种类型

string / hash / list /set /zset

List 分页第一页 热数据 数据库中经常查询的数据 很少修改的数据

Set 去重数据

Zset 带排序的权重数据

服务端

redis-server.exe redis.windows.conf

客户端

redis-cli -h 127.0.0.2 -p 6379 -a 123456

启动客户端显示中文字体：

redis-cli -h 127.0.0.1 -p 6379 -a 123456 --raw

Redis 集群搭建最少6个节点

主从复制+分片

主从复制：当一个集群中的主节点挂了，则有从节点代替

分片：解决所有的读取不会所有的数据读取一台集群或者一个主节点

启动集群指令

Linux ./redis-trib.rb create --replicas 1 127.0.0.2:6379 127.0.0.2:6380 127.0.0.2:6381 127.0.0.2:6382 127.0.0.2:6383 127.0.0.2:6384

Windows redis-trib.rb create --replicas 1 127.0.0.2:6379 127.0.0.2:6380 127.0.0.2:6381 127.0.0.2:6382 127.0.0.2:6383 127.0.0.2:6384

注意127.0.0.2 ip时失败

redis-trib.rb create --replicas 1 127.0.0.1:6379 127.0.0.1:6380 127.0.0.1:6381 127.0.0.1:6382 127.0.0.1:6383 127.0.0.1:6384

客户端

redis-cli -h 127.0.0.1 -c -p 6379 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6380 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6381 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6382 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6383 -a 123456

Redis-cli -h 127.0.0.1 -c -p 6384 -a 123456

Flushall

Cluster reset

redis-trib.rb check 127.0.0.1:6379

官方是推荐使用redis-trib.rb fix 来修复集群。通过cluster nodes看到7001这个节点被干掉了。可以按照下面操作进行修复

[root@slave2 redis]# redis-trib.rb fix 127.0.0.2:6380

修复完成后再用check命令检查下是否正确（查看别的节点）

[root@slave2 redis]# redis-trib.rb check 127.0.0.2:6380

只要输入任意集群中节点即可，会自动检查所有相关节点。

可以查看相应的输出看下是否是每个Master都有了slots。

如果分布不均匀那可以使用下面的方式重新分配slot:

[root@slave2 redis]# redis-trib.rb reshard 127.0.0.2:6379

集群搭建遇到问题

port 7000 //端口7000,7002,7003

bind 本机ip //默认ip为127.0.0.1 需要改为其他节点机器可访问的ip 否则创建集群时无法访问对应的端口，无法创建集群

daemonize yes //redis后台运行

pidfile /var/run/redis\_7000.pid //pidfile文件对应7000,7001,7002

cluster-enabled yes //开启集群 把注释#去掉

cluster-config-file nodes\_7000.conf //集群的配置 配置文件首次启动自动生成 7000,7001,7002

cluster-node-timeout 15000 //请求超时 默认15秒，可自行设置

appendonly yes //aof日志开启 有需要就开启，它会每次写操作都记录一条日志

syslog-ident = redis  
Invalid argument during startup: unknown conf file parameter : cluster-config-file

我一开始以为是redis的windows服务版本不支持这个配置选项,网上各种找和查,最后才知道原来redis的配置项前面不能有空格

关于启动redis集群时：

**[ERR] Node 172.168.63.202:7001 is not empty. Either the nodealready knows other nodes (check with CLUSTER NODES) or contains some**

的问题

解决办法：

1)将每个节点下aof、rdb、nodes.conf本地备份文件删除；   
2)172.168.63.201:7001> flushdb #清空当前数据库(可省略)

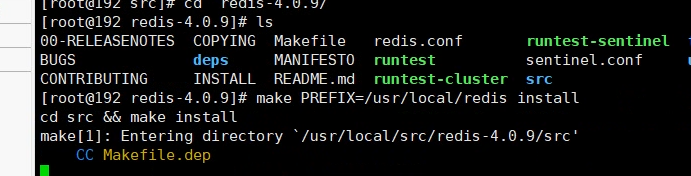
dotnet ConsoleAppServer.dll --ip=127.0.0.1 --port=998

秒杀测试指令

dotnet ConsoleAppNetCore3.dll 1 56

Redis Lunix安装

make PREFIX=/usr/local/redis install



cd /usr/local/redis/bin/

./redis-server redis.conf

./redis-cli -h 127.0.0.1 -p 6379

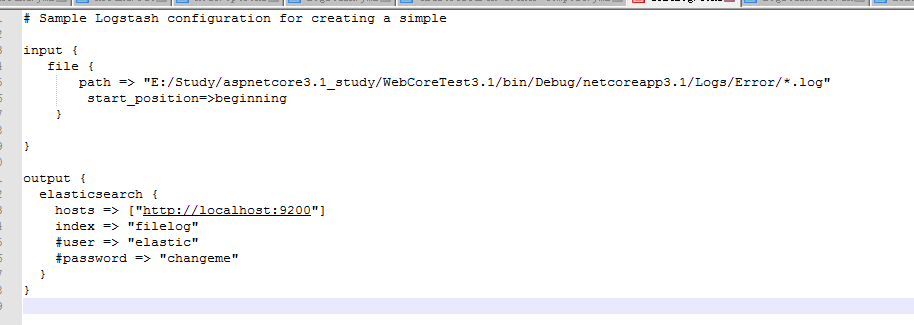
# ELK Elasticsearch Logstash Kibana

一、Logstash 启动配置：

1、指令：logstash.bat -f filelog.conf

Filelog.conf配置内容

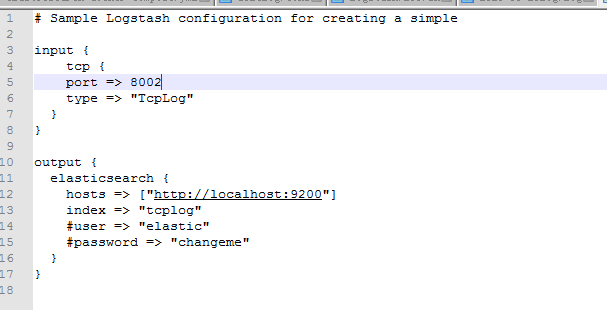
注意点：path路径斜线是 “/” 不是 “\”



2、Tcplog.conf配置内容

dotnet WebCoreTest\_ELK\_Logstash\_TCP.dll --urls=http://\*:8003 --ip=127.0.0.1 --port=8003

logstash.bat -f tcplog.conf

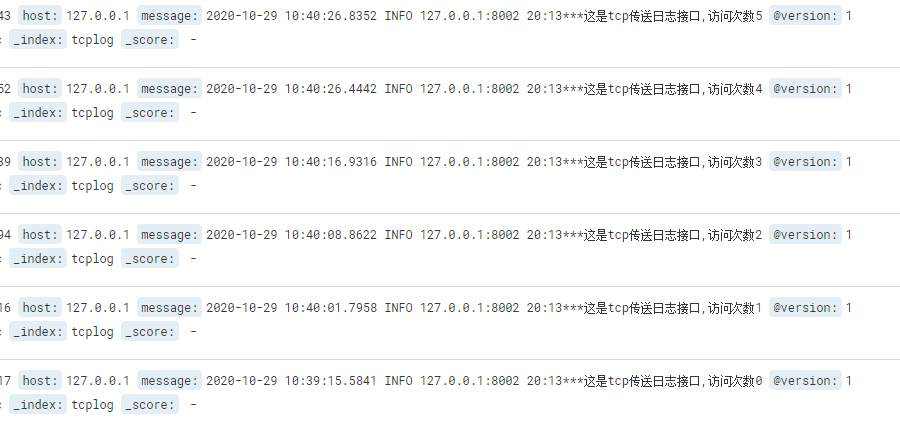


注意点：core项目配置nlog.config时配置tcp协议

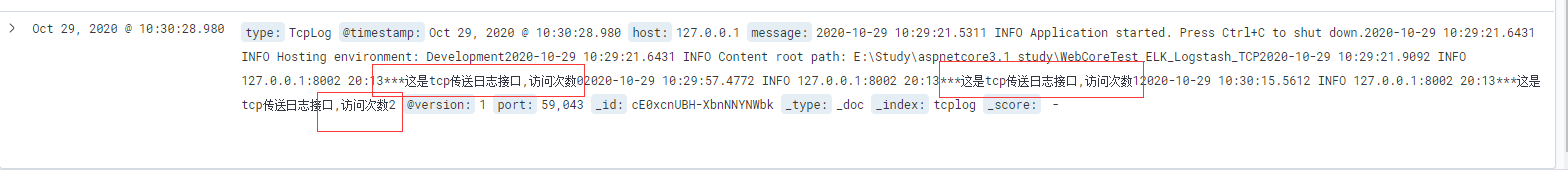
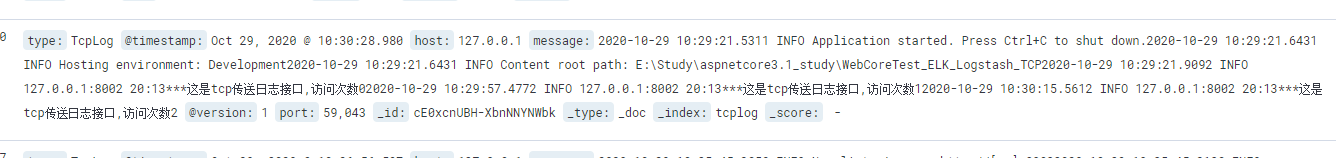
<target xsi:type="Network" name="logInfo" address="tcp://127.0.0.1:8002" layout="${longdate} ${uppercase:${level}} ${message}" keepConnection="false"/>



当属性keepConnection="false"，可以持续写入，



当设置true时，只能关闭应用后一次性写入，我第一次采坑点就是没有配置<target xsi:type="Network" name="logInfo" address="tcp://127.0.0.1:8002" layout="${longdate} ${uppercase:${level}} ${message}" />导致只能关闭应用时候一次性写入。我个人猜测是tcp端口持续暂用导致，估计八九不离十（我猜测是在本地运行core项目中应该是只能写进去一次，8002被占用，logstash应该调用不到，导致日志暂时写不进去，当core项目停止后，logstash能通过tcp传送成功）

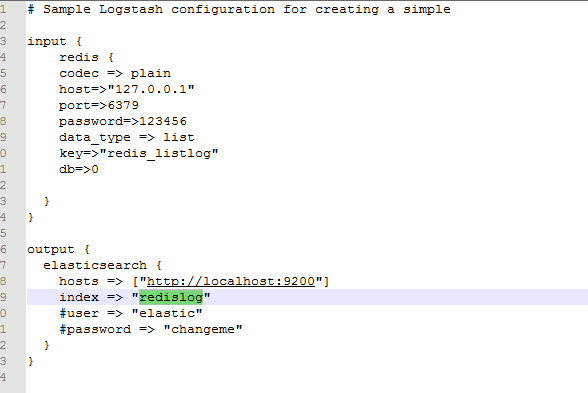


3、Redis 方式写入

logstash.bat -f redislog.conf

配置好之后按照顺序重启：redis-- es ---logstash--kibana 客户端写入正常拉取

（大数据高并发情况下日志可以采取存储到redis list中，然后配置logstash去拉取数据，logstash拉取数据后会删除，所以不用担心内存不足问题

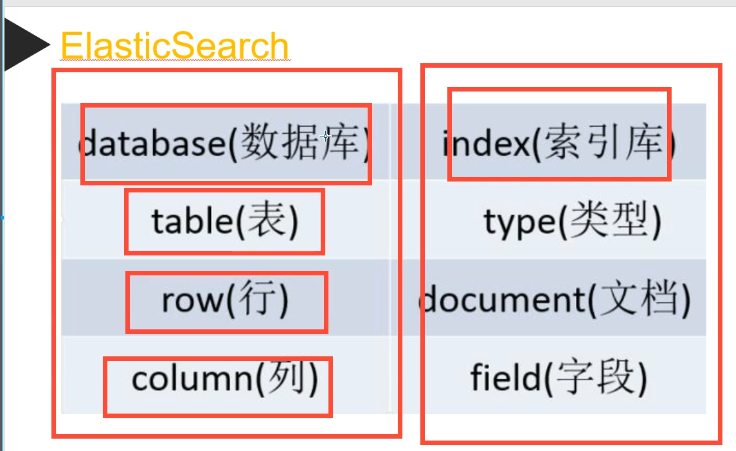
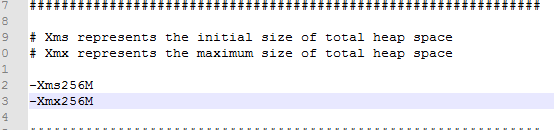


### ElasticSearch

Elasticsearch配置：

Config/jvm.options:

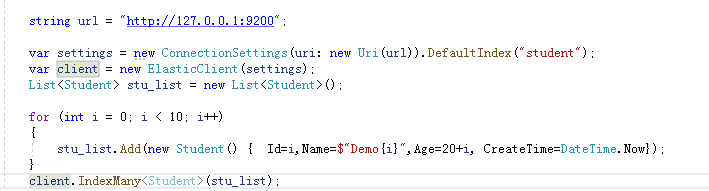
-Xms/Xmx默认是1g,改成256M



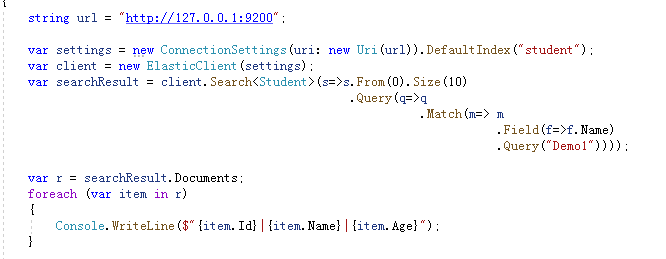
.NetCore调用需要引用NEST包

官方文档：https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/index.html

插入es数据库代码



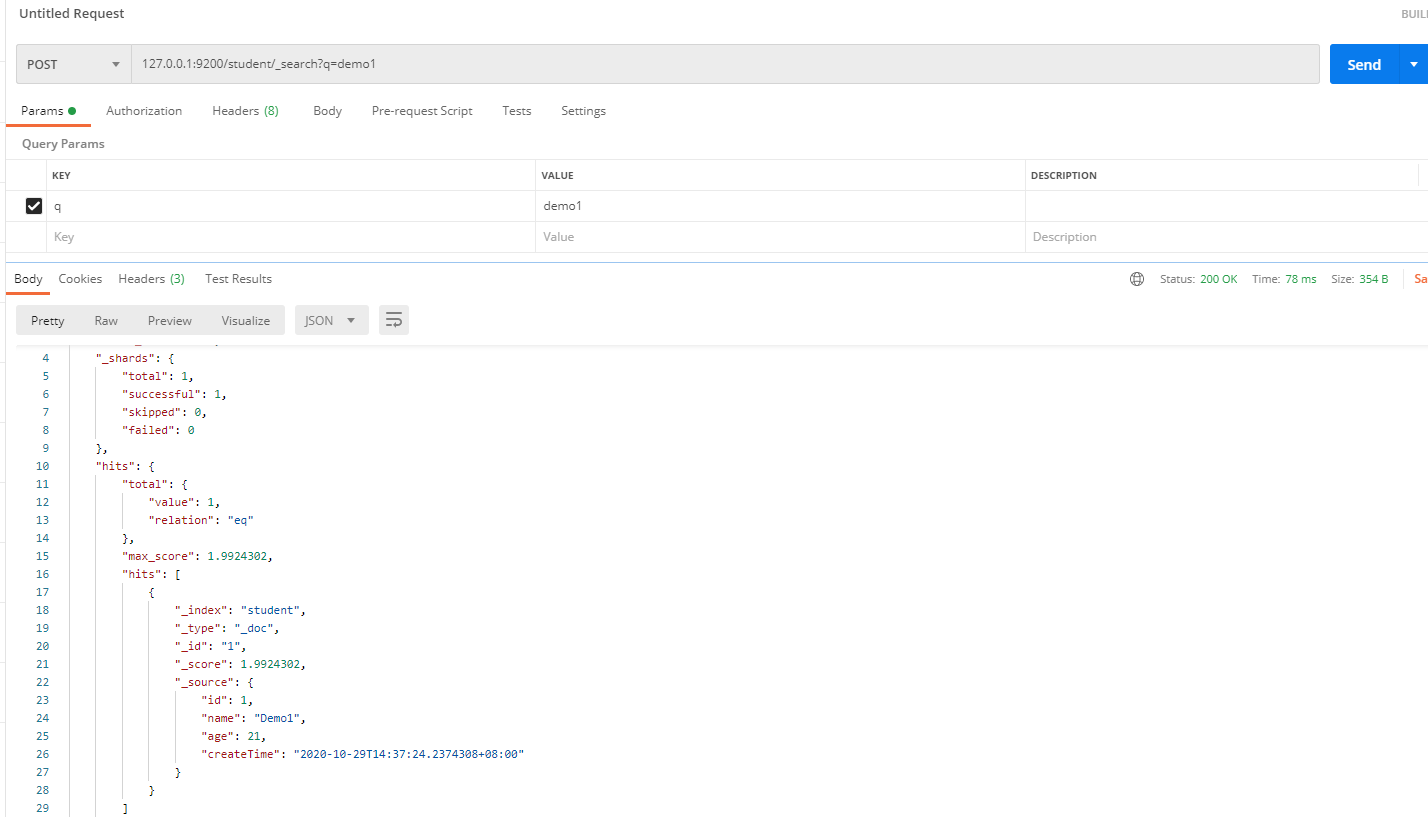
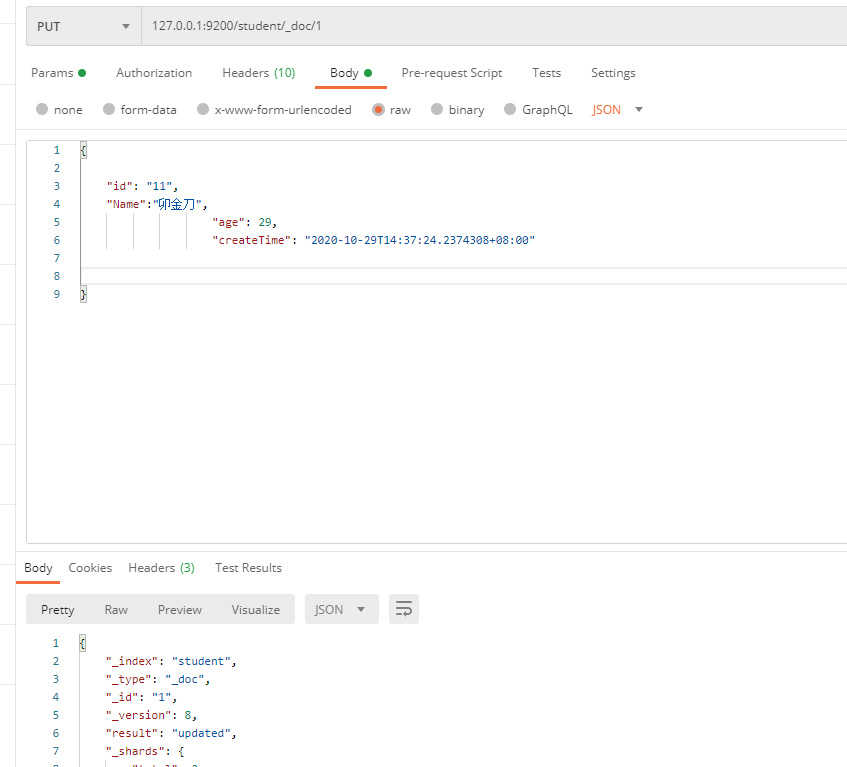
获取代码：



Postman操作

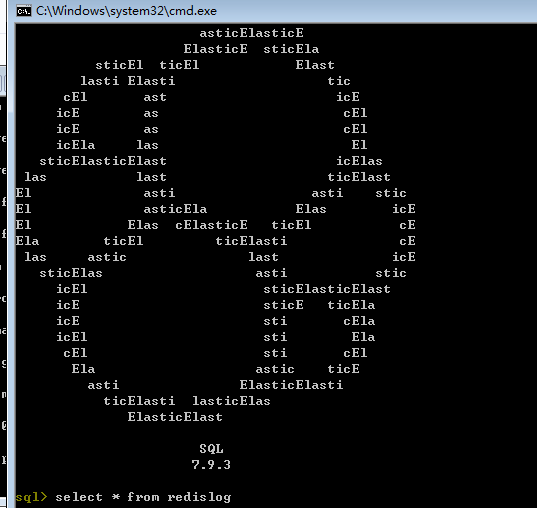
此篇博客可以参考https://blog.csdn.net/u013613428/article/details/56484794

官网文档https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html



使用sql操作

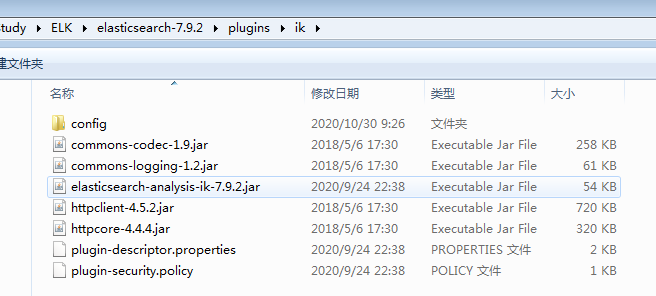
Bin/elasticsearch-sql-cli.bat



IK分词器（插件中文分词）

\elasticsearch-analysis-ik-7.9.2\bin\文件加下的分词器文件copy放到

elasticsearch下面的一个\plugins文件夹下建立一个新的文件夹\ik\文件夹下



案例：

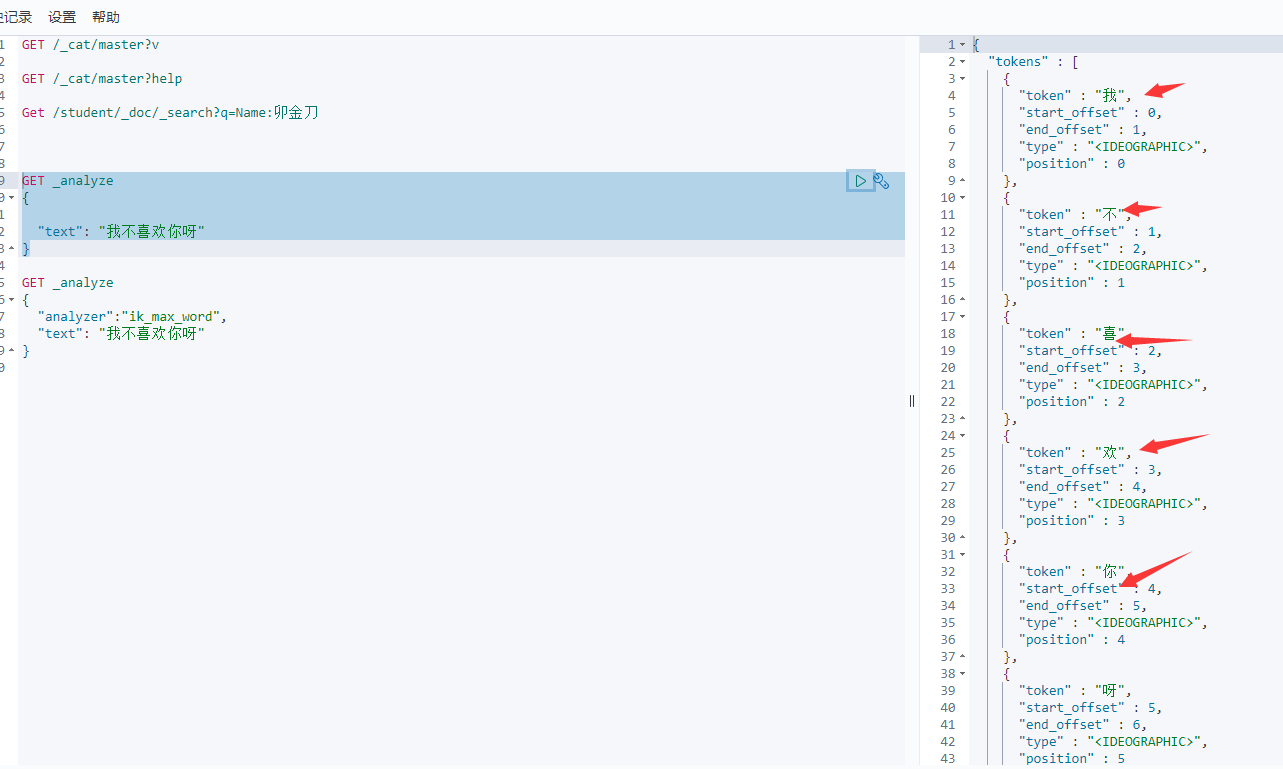
未使用分词器：

GET \_analyze

{

"text": "我不喜欢你呀"

}



使用分词器：

GET \_analyze

{

"analyzer":"ik\_max\_word",

"text": "我不喜欢你呀"

}



1. Kibana

使用前先安装java环境包

查询操作

Get /student/\_doc/\_search

Get /student/\_doc/\_search?q=Name:卯金刀

# Docker

命令：

docker --version 查看当前版本

docker ps 查看当前运行的容器

docker images 查看本地所有的镜像

docker run 运行一个容器

docker pull 用来拉取一个镜像到本地

1 docker logs 容器名称

2 docker attach 容器名称或容器ID

3 docker exec -it 59ec /bin/bash (如果提示“/bin/bash 未找到”之类的 可以改成:docker exec -it 59ec sh,退出ctrl+d)

4 docker stop 容器ID或容器名

5 docker kill 容器ID或容器名

6 docker rm 容器ID或容器名

7 systemctl restart docker  
8 docker image prune 删除所有镜像  
9.docker container prune 删除所有容器

运行docker容器，后台运行

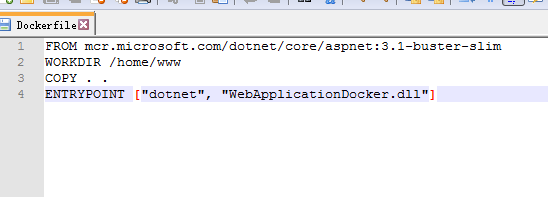
docker run -d -p 80:80 webapicore

docker stop 容器id

docker rm 容器id

部署CORE

1、上传Dockerfile文件



1. 项目上传后执行

cd /home/apitest/ 进入该项目目录

docker build -t webapicore -f Dockerfile . 注意一个英文的“.”不能少

docker build -t webmvccore -f Dockerfile .

1. docker run -d -p 5026:80 webapicore
2. docker run -d -p 5027:80 webapicore

docker run -d -p 8001:8001 webapicore

1. docker run -d -p 8001:80 webmvccore

查看端口监听情况

netstat -an | grep 8080

查找文件路径find / -name nginx.conf

查找文件夹 ls -l /usr/share/nginx/

编辑文件

vim nginx.conf

创建文件夹

mkdir filesname

nginx 文件挂载

docker run --name nginx -p 8086:80 -v /data/docker\_v/nginx/www:/home/configfile -d nginx

退出：先按【ESC】 再输入:q

常用操作指令http://www.51gjie.com/linux/1029.html

批量停用并删除全部容器

1.停用全部运行中的容器:

docker stop $(docker ps -q)

2.删除全部容器：

docker rm $(docker ps -aq)

3.一条命令实现停用并删除容器：

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

docker-compose命令安装镜像：

1、运行以下命令以下载Docker Compose的当前稳定版本：

sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2将可执行权限应用于二进制文件：  
sudo chmod +x /usr/local/bin/docker-compose

3测试安装。  
docker-compose --version



1. 执行docker-compose命令

a先启动consul集群

cd /home/configfile/

docker-compose up

b在启动网站

cd /home/project/

docker-compose up

consule

拉取最新镜像：

docker pull consule

启动consul:

docker run -d -p 11500:8500 consul

consul 搭建集群

搭建集群前需要开通ip端口：tcp :8301,8302,8500,9500,10500,11500/udp:8301,8302

node1:

docker run -d --name=node1 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 8300:8300 \

-p 8301:8301 \

-p 8301:8301/udp \

-p 8302:8303 \

-p 8302:8302/udp \

-p 8400:8400 \

-p 8500:8500 \

-p 8600:8600 \

-h node1 \

consul agent -server -bootstrap-expect=3 -bind=0.0.0.0 -node=node1 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

node2

docker run -d --name=node2 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 9300:8300 \

-p 9301:8301 \

-p 9301:8301/udp \

-p 9302:8303 \

-p 9302:8302/udp \

-p 9400:8400 \

-p 9500:8500 \

-p 9600:8600 \

-h node2 \

consul agent -server -bind=0.0.0.0 \

-join=121.196.176.108 -node-id=$(uuidgen | awk '{print tolower($0)}') \

-node=node2 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

node3

docker run -d --name=node3 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 10300:8300 \

-p 10301:8301 \

-p 10301:8301/udp \

-p 10302:8303 \

-p 10302:8302/udp \

-p 10400:8400 \

-p 10500:8500 \

-p 10600:8600 \

-h node3 \

consul agent -server -bind=0.0.0.0 \

-join=121.196.176.108 -node-id=$(uuidgen | awk '{print tolower($0)}') \

-node=node3 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

node4

docker run -d --name=node4 --restart=always \

-e ’CONSUL\_LOCAL\_CONFIG={“skip\_leave\_on\_interrupt”:true}’ \

-p 11300:8300 \

-p 11301:8301 \

-p 11301:8301/udp \

-p 11302:8303 \

-p 11302:8302/udp \

-p 11400:8400 \

-p 11500:8500 \

-p 11600:8600 \

-h node4 \

consul agent -bind=0.0.0.0 \

-join=121.196.176.108 -node-id=$(uuidgen | awk '{print tolower($0)}') \

-node=node4 \

-data-dir=/tmp/data-dir -client 0.0.0.0 -ui

查看日志

docker logs -f node1

查看集群节点

docker exec -t node1 consul members

docker exec -t consul1 consul members

查看主从信息

docker exec -t node1 consul operator raft list-peers

docker exec -t consul1 consul operator raft list-peers

Nginx

nginx 挂载转发到consul集群

docker run -d -p 8089:80 -v /home/project/consulnginx/:/var/log/nginx/ -v /home/project/consulnginx/nginx.conf:/etc/nginx/nginx.conf --name consulnginx nginx

docker run -d -p 8088:80 -v /home/project/gatewaynginx/:/var/log/nginx/ -v /home/project/gatewaynginx/nginx.conf:/etc/nginx/nginx.conf --name gatewaynginx nginx

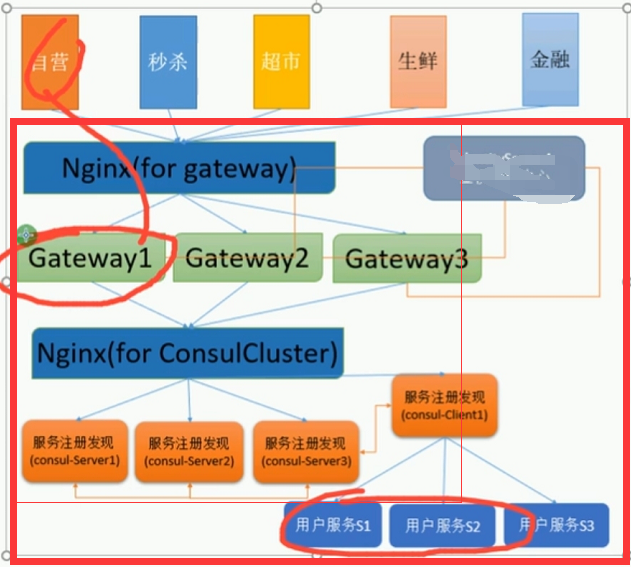
先启动consul集群，再启动项目

到此本次搭建的框架图：

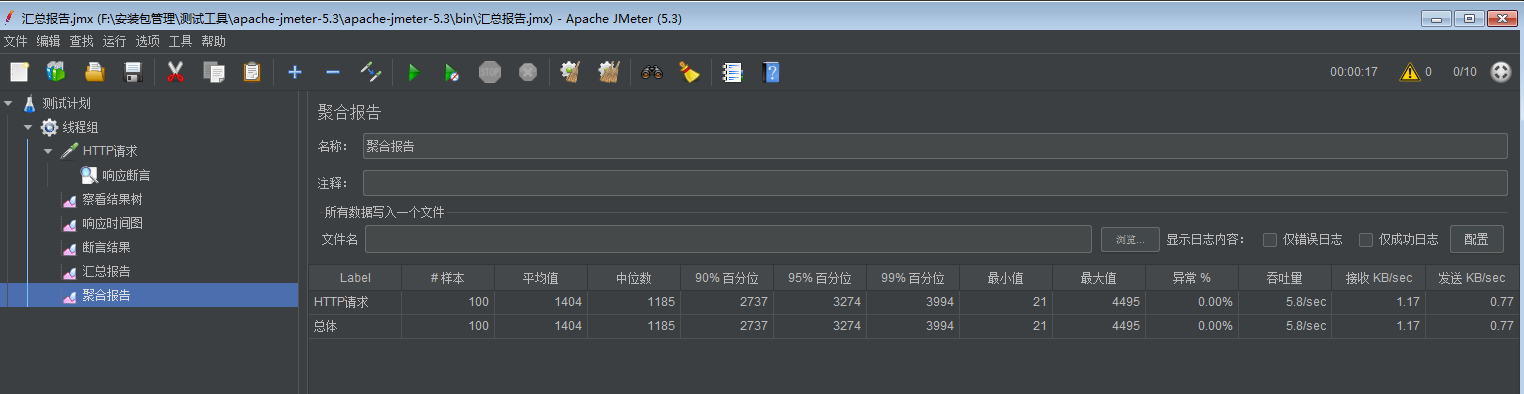
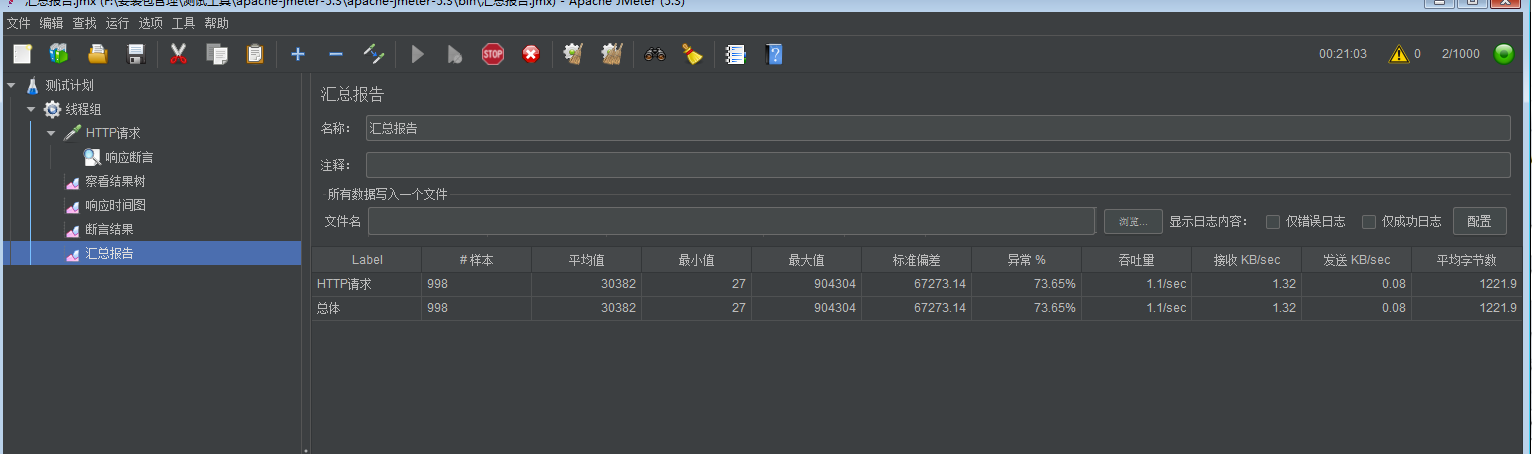
客户端— —》Nginx（gateway网关）——》网关（集群）——》Nginx（consulf服务发现与治理）——》consul(集群)——》webapi服务接口（集群）

：

|  |  |  |  |
| --- | --- | --- | --- |
|  | IP：Ports | 调用服务(请求网关) |  |
| 客户端 | 121.196.176.108:xxx | http://121.196.176.108:8088/pserver/products | http://121.196.176.108:8088/userver/user |
| Nginx（gateway网关） | 121.196.176.108:8088 |  |  |
| 网关（集群） | 121.196.176.108:6299/6399/6499 |  |  |
| Nginx（consulf服务发现与治理） | 121.196.176.108:8089 |  |  |
| consul(集群) | 121.196.176.108:8500/9500/10500/11500 |  |  |
| webapi\_user服务接口（集群） | 121.196.176.108:5026/5027/5028 |  |  |
| webapi\_product服务接口（集群） | 121.196.176.108:6026/6027/6028 |  |  |



JMeter 压测工具



解释一下这份报告：

样本：请求的次数，计算公式是线程数\*循环次数，如果线程组配置勾选了永远，那么就是你停止测试时实际发送的请求数

平均值：响应时间的平均用时，单位是毫秒。比如这里的平均响应时间是38毫秒

中位数：响应时间的中位数，单位是毫秒。

90%百分位：90%的响应时间小于该数值，单位是毫秒。这里有90%的响应时间小于22毫秒

95%百分位：含义和90%类似

99%百分位：含义和90%类似

最小值：本轮测试最小响应时间，单位是毫秒。

最大值：本轮测试最大响应时间，单位是毫秒。

异常%：本轮测试出现异常的请求比例。

吞吐量：可以理解为QPS，即是我们测试的接口处理请求的能力。比如这里是平均每秒可以处理2.2次请求

接收KB/Sec：响应数据的接收速率

发送KB/Sec：请求数据的发送速率

数据库：

1. 分库分表分区

mysql 分区性能优化

SQLserver分区性能优化：

创建分区数据库

create database testslcake on primary

(

name=N'Sales',

filename=N'E:\Study\data\slcake\testslcake.mdf',

size=5MB,

maxsize=100MB,

filegrowth=10%

),

filegroup FG1

(

NAME = N'File1',

FILENAME = N'E:\Study\data\slcake\FG1\_File.ndf',

SIZE = 1MB,

MAXSIZE = 100MB,

FILEGROWTH = 10%

),

FILEGROUP FG2

(

NAME = N'File2',

FILENAME = N'E:\Study\data\slcake\FG2\_File.ndf',

SIZE = 1MB,

MAXSIZE = 100MB,

FILEGROWTH = 10%

),

FILEGROUP FG3

(

NAME = N'File3',

FILENAME = N'E:\Study\data\slcake\FG3\_File.ndf',

SIZE = 1MB,

MAXSIZE = 100MB,

FILEGROWTH = 10%

) ,

FILEGROUP FG4

(

NAME = N'File4',

FILENAME = N'E:\Study\data\slcake\FG4\_File.ndf',

SIZE = 1MB,

MAXSIZE = 100MB,

FILEGROWTH = 10%

)

LOG ON

(

NAME = N'testslcake\_Log',

FILENAME = N'E:\Study\data\slcake\testslcake\_Log.ldf',

SIZE = 1MB,

MAXSIZE = 100MB,

FILEGROWTH = 10%

)

GO

分区函数

CREATE PARTITION FUNCTION pf\_OrderTime (datetime)

AS RANGE RIGHT

FOR VALUES ('2018/01/01', '2019/01/01','2020/01/01')

GO

分区方案

create partition scheme ps\_OrderTime

as partition pf\_OrderTime

to(FG1,FG2,FG3,FG4)

GO

查询分区文件中包含多少数据

select $partition.pf\_OrderTime(OrderTime) as Patition,COUNT(\*) countRow from dbo.[ORDER]

group by $partition.pf\_OrderTime(OrderTime)

SELECT \* FROM [order] WHERE $partition.pf\_OrderTime(OrderTime)=4

Git操作

…or create a new repository on the command line  
echo "# suppermall" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin https://github.com/fangliu520/suppermall.git  
git push -u origin main  
   
…or push an existing repository from the command line  
git remote add origin https://github.com/fangliu520/suppermall.git  
git branch -M main  
git push -u origin main

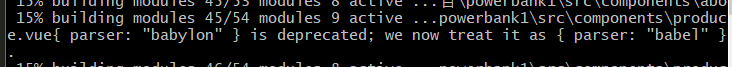
webpack 打包工具

命令：

npm run build

npm install

**[在vue项目中，执行 npm run dev 时提示 { parser: "babylon" } is deprecated; we now treat it as { parser: "babel" }](https://www.cnblogs.com/ruilin/p/11177873.html)**



这个不是报错，而是需要将babylon 换成babel

解决办法是：

找到modules包里面的：node\_modules\vue-loader\lib\template-compiler\index.js

将{ parser: "babylon" } 换成?{ parser: "babel" } 即可；

如下图所示：



"devDependencies": {

"babel-core": "^6.26.3",

"babel-loader": "^7.1.5",

"babel-preset-es2015": "^6.24.1",

"css-loader": "^3.6.0",

"file-loader": "^4.0.0",

"style-loader": "^1.3.0",

"url-loader": "^4.1.1",

"vue-loader": "^13.0.0",

"vue-template-compiler": "^2.6.12",

"webpack": "^3.6.0"

},

<!--安装webpack打包工具

cmd> npm install webpack@3.6.0-->

<!--通过webpack 打包工具

webpack ./layuistudy/src/main.js ./layuistudy/dist/index.js-->

<!--npm run build-->

<!--es6语法转es5 安装包 : npm install --save-dev babel-loader@7 babel-core babel-preset-es2015 -->

安装支持vue文件

npm install --save-dev vue-loader vue-template-compiler

修改webpack.config.js文件中plugins内容

{

test: /\.vue$/,

loader: 'vue-loader'

}

resolve: {

alias: {

'vue$': 'vue/dist/vue.esm.js' // 用 webpack 1 时需用 'vue/dist/vue.common.js'

},

extensions:['.js','.css','.vue']

}



<!--支持打包vue文件 ： -->

安装HtmlWebpackPlugin 插件

npm install --save-dev html-webpack-plugin

修改package.json

"html-webpack-plugin": "^3.0.0",

修改webpack.config.js文件中plugins内容

const htmlWebpackPlugin = require('html-webpack-plugin');

plugins: [

new htmlWebpackPlugin({

template:'index.html'

})

]

安装JS压缩uglifyjs-webpack-plugin 插件

npm install --save-dev uglifyjs-webpack-plugin@1.3.0

修改package.json

"uglifyjs-webpack-plugin": "^1.1.1",

修改webpack.config.js文件中plugins内容

const uglifyjsWebpackPlugin=require('uglifyjs-webpack-plugin');

new uglifyjsWebpackPlugin()

安装webpack本地服务器

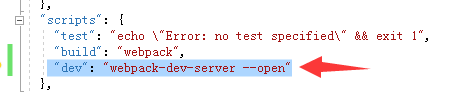
这个本地服务基于node.js搭建，内部使用express框架，可以实现我们想要的让浏览器自动刷新显示我们修改后的效果

npm install --save-dev webpack-dev-server

npm install --save-dev [webpack-dev-server@2.9.1](mailto:webpack-dev-server@2.9.1)

修改package.json

"dev": "webpack-dev-server --open"



修改webpack.config.js文件中plugins内容

devServer: {

contentBase: './dist',

inline: true,

port:9090

}

运行命令

npm run dev

安装webpack-merge 拆分生产、本地环境配置

npm install --save-dev webpack-merge

*开发*和*生产*版本的目标差异很大。在*开发中*，我们需要强大的源映射和具有实时重载或热模块替换的localhost服务器。在*生产中*，我们的目标转向着眼于缩小包装，减轻重量源图以及优化资产以缩短装载时间。考虑到这种逻辑上的分离，我们通常建议为每种环境编写**单独的Webpack配置**。

虽然我们将*生产*和*开发*专用的部分分开，但请注意，我们仍将保持“通用”配置以保持干燥。为了将这些配置合并在一起，我们将使用一个名为的实用程序[webpack-merge](https://github.com/survivejs/webpack-merge)。有了“通用”配置，我们就不必在特定于环境的配置中重复代码。

让我们从安装webpack-merge和拆分之前指南中已经开始的工作开始：

"build": "webpack --config ./build/prod.config.js",

"dev": "webpack-dev-server --open --config ./build/dev.config.js"

**[实例生命周期钩子](https://cn.vuejs.org/v2/guide/instance.html#实例生命周期钩子)**

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会。

比如 **[created](https://cn.vuejs.org/v2/api/#created)** 钩子可以用来在一个实例被创建之后执行代码：

new Vue({

data: {

a: 1

},

created: function () {

// `this` 指向 vm 实例

console.log('a is: ' + this.a)

}

})

// => "a is: 1"

也有一些其它的钩子，在实例生命周期的不同阶段被调用，如 **[mounted](https://cn.vuejs.org/v2/api/#mounted)**、**[updated](https://cn.vuejs.org/v2/api/#updated)** 和 **[destroyed](https://cn.vuejs.org/v2/api/#destroyed)**。生命周期钩子的 this 上下文指向调用它的 Vue 实例。

不要在选项 property 或回调上使用**[箭头函数](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow_functions)**，比如 created: () => console.log(this.a) 或 vm.$watch('a', newValue => this.myMethod())。因为箭头函数并没有 this，this 会作为变量一直向上级词法作用域查找，直至找到为止，经常导致 Uncaught TypeError: Cannot read property of undefined 或 Uncaught TypeError: this.myMethod is not a function 之类的错误。

**[生命周期图示](https://cn.vuejs.org/v2/guide/instance.html#生命周期图示)**

下图展示了实例的生命周期。你不需要立马弄明白所有的东西，不过随着你的不断学习和使用，它的参考价值会越来越高。

Vue声明周期



安装 Vue CLI脚手架

npm install -g @vue/cli  
# OR  
yarn global add @vue/cli

npm install -g @vue/cli

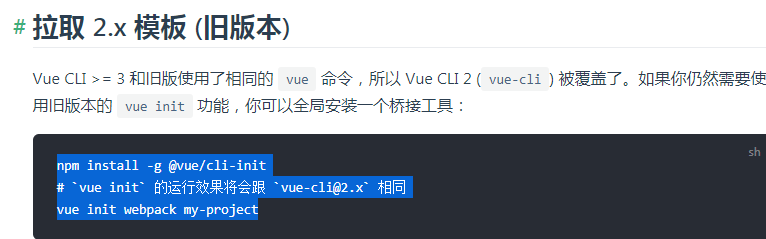
vue --version

升级

npm update -g @vue/cli

创建项目：

vue create hello-world



npm install -g @vue/cli-init  
# `vue init` 的运行效果将会跟 `vue-cli@2.x` 相同

创建vuecli2.x项目：

vue init webpack my-project

vue模版解析过程：

构建webpack项目的选择：runtime-compiler

template->ast(abstract syntax tree)->render->vdom->UI

选择runtime-only

render->vdom->UI

vuecli2.x运行：

npm run build

npm run dev

npm start

vuecli3运行：

npm run serve

npm run build

Vue项目管理器界面

命令：

vue ui



箭头函数：

格式：

()=>{}

//一个参数简写

//const squer=(num)=>{return num\*num;}

const squer = num => num \* num;

console.log(squer(10));

//两个参数

const mul = (num1, num2) => num1 \* num2;

console.log(mul(10,20));

箭头函数中的this:

结论：箭头函数this引用的就是最近作用域中的this

向外层作用域中，一层层查找this,直到有this 的定义

const obj = {

aa() {

setTimeout(function () {

console.log("普通函数",this);//this输出window对象

}, 1000);

setTimeout(() => {

console.log("箭头函数",this)//this输出obj对象

},1000)

}

}

obj.aa();



const obj = {

aa() {

setTimeout(function () {

setTimeout(function () {

console.log("普通函数", this);//this输出window对象

});

setTimeout(() => {

console.log("普通函数中箭头函数", this);//this输出window对象

})

},1000);

setTimeout(() => {

setTimeout(function () {

console.log("箭头函数中普通函数", this);//this输出window对象

});

setTimeout(() => {

console.log("箭头函数", this)//this输出obj对象

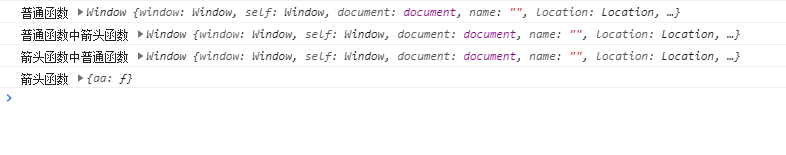
})

},1000)

}

}

obj.aa();



vue-router 路由

npm install vue-router --save

vuex状态管理

npm install vuex --save

state,mutations,getters,actions,modules



axios网络请求安装

npm install axios --save

<http://www.axios-js.com/zh-cn/docs/>

执行多个并发请求  
  
function getUserAccount() {  
 return axios.get('/user/12345');  
}  
  
function getUserPermissions() {  
 return axios.get('/user/12345/permissions');  
}  
  
axios.all([getUserAccount(), getUserPermissions()])  
 .then(axios.spread(function (acct, perms) {  
 // 两个请求现在都执行完成  
 }));

可以通过向 axios 传递相关配置来创建请求  
  
axios(config)  
// 发送 POST 请求  
axios({  
 method: 'post',  
 url: '/user/12345',  
 data: {  
 firstName: 'Fred',  
 lastName: 'Flintstone'  
 }  
}).then(res=>{

结果

});

better-scroll滚动框架

Toast(吐司)封装

弹出框通用封装

1. 普通封装 （已经实现）
2. 插件封装 (暂时出现vue.extend 未定义错误)

fastClick 减少点击延迟

安装fastclick

npm install fastclick --save

在main.js中加入

//解决移动端300ms延迟问题

import FastClick from 'fastclick'

FastClick.attach(document.body);

图片懒加载

https://github.com/hilongjw/vue-lazyload

npm install vue-lazyload --save

postcss-px-to-viewport 将px单位转换成vm插件

npm install postcss-px-to-viewport --save-dev

安装完成后在vue根目录下找到".postcssrc.js"文件，添加下面的配置：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | "postcss-px-to-viewport":{  viewportWidth: 750, // 视窗的宽度，对应的是我们设计稿的宽度，一般是750.  viewportHeight: 1334, // 视窗的高度，根据750设备的宽度来指定，一般指定1334.  unitPrecision: 3, // (指定`px`转换为视窗单位值的小数位数（很多时候无法整除).  viewportUnit: 'vw', // 指定需要转换成的视窗单位，建议使用vw.  selectorBlackList: ['.ignore', '.hairlines','weui'], // 指定不转换为视窗单位的类，可以自定义，可以无限添加,建议定义一至两个通用的类名.  minPixelValue: 1, // 小于或等于`1px`不转换为视窗单位，你也可以设置为你想要的值.  mediaQuery: false // 允许在媒体查询中转换`px`.  } |